Apache Spark - Practice set 4

Total points 41/60

The respondent's email (**saurav.karki@fusemachines.com**) was recorded on submission of this form.

✓ Which of the following options describes the responsibility of the executors in Spark? 1/1

- ○ The executors accept jobs from the driver, analyze those jobs, and return results to the driver.
- ○ The executors accept tasks from the driver, execute those tasks, and return results to the cluster manager.
- ● The executors accept tasks from the driver, execute those tasks, and return results to the driver. ✓
- ○ The executors accept tasks from the cluster manager, execute those tasks, and return results to the driver.
- ○ The executors accept jobs from the driver, plan those jobs, and return results to the cluster manager.

✓ Which of the following describes the role of tasks in the Spark execution hierarchy? 1/1

- ● Tasks are the smallest element in the execution hierarchy. ✓
- ○ Within one task, the slots are the unit of work done for each partition of the data.
- ○ Tasks are the second-smallest element in the execution hierarchy.
- ○ Stages with narrow dependencies can be grouped into one task.
- ○ Tasks with wide dependencies can be grouped into one stage.

✗ Which of the following describes the role of the cluster manager? 0/1

- ○ The cluster manager schedules tasks on the cluster in client mode.
- ○ The cluster manager schedules tasks on the cluster in local mode.
- ○ The cluster manager allocates resources to Spark applications and maintains the executor processes in client mode.
- ● The cluster manager allocates resources to Spark applications and maintains the executor processes in remote mode. ✗
- ○ The cluster manager allocates resources to the DataFrame manager.

Correct answer

- ● The cluster manager allocates resources to Spark applications and maintains the executor processes in client mode.

✓ Which of the following is the idea behind dynamic partition pruning in Spark? 1/1

- ● Dynamic partition pruning is intended to skip over the data you do not need in the results of a query. ✓
- ○ Dynamic partition pruning concatenates columns of similar data types to optimize join performance.
- ○ Dynamic partition pruning performs wide transformations on disk instead of in memory.
- ○ Dynamic partition pruning reoptimizes physical plans based on data types and broadcast variables.
- ○ Dynamic partition pruning reoptimizes query plans based on runtime statistics collected during query execution.

✓ Which of the following is one of the big performance advantages that Spark has over Hadoop? 1/1

- Spark achieves great performance by storing data in the DAG format, whereas Hadoop can only use parquet files.
- Spark achieves higher resiliency for queries since, different from Hadoop, it can be deployed on Kubernetes.
- [✓ Spark achieves great performance by storing data and performing computation in memory, whereas large jobs in Hadoop require a large amount of relatively slow disk I/O operations.] ✓
- Spark achieves great performance by storing data in the HDFS format, whereas Hadoop can only use parquet files.
- Spark achieves performance gains for developers by extending Hadoop's DataFrames with a user-friendly API.

✓ Which of the following is the deepest level in Spark's execution hierarchy? 1/1

- Job
- [✓ Task] ✓
- Executor
- Slot
- Stage

❌ Which of the following statements about garbage collection in Spark is incorrect? 0/1

- ○ Garbage collection information can be accessed in the Spark UI's stage detail view.
- ○ Optimizing garbage collection performance in Spark may limit caching ability.
- ○ Manually persisting RDDs in Spark prevents them from being garbage collected.
- ○ In Spark, using the G1 garbage collector is an alternative to using the default Parallel garbage collector.
- ● Serialized caching is a strategy to increase the performance of garbage collection. ❌

Correct answer

- ● Manually persisting RDDs in Spark prevents them from being garbage collected.

❌ Which of the following describes characteristics of the Dataset API? 0/1

- ○ The Dataset API does not support unstructured data.
- ● In Python, the Dataset API mainly resembles Pandas' DataFrame API. ❌
- ○ In Python, the Dataset API's schema is constructed via type hints.
- ○ The Dataset API is available in Scala, but it is not available in Python.
- ○ The Dataset API does not provide compile-time type safety.

Correct answer

- ● The Dataset API is available in Scala, but it is not available in Python.

✗ Which of the following describes the difference between client and cluster execution modes? 0/1

- ○ In cluster mode, the driver runs on the worker nodes, while the client mode runs the driver on the client machine.
- ○ In cluster mode, the driver runs on the edge node, while the client mode runs the driver in a worker node.
- [✓ In cluster mode, each node will launch its own executor, while in client mode, executors will exclusively run on the client machine.] ✗
- ○ In client mode, the cluster manager runs on the same host as the driver, while in cluster mode, the cluster manager runs on a separate node.
- ○ In cluster mode, the driver runs on the master node, while in client mode, the driver runs on a virtual machine in the cloud.

Correct answer

- [✓ In cluster mode, the driver runs on the worker nodes, while the client mode runs the driver on the client machine.]

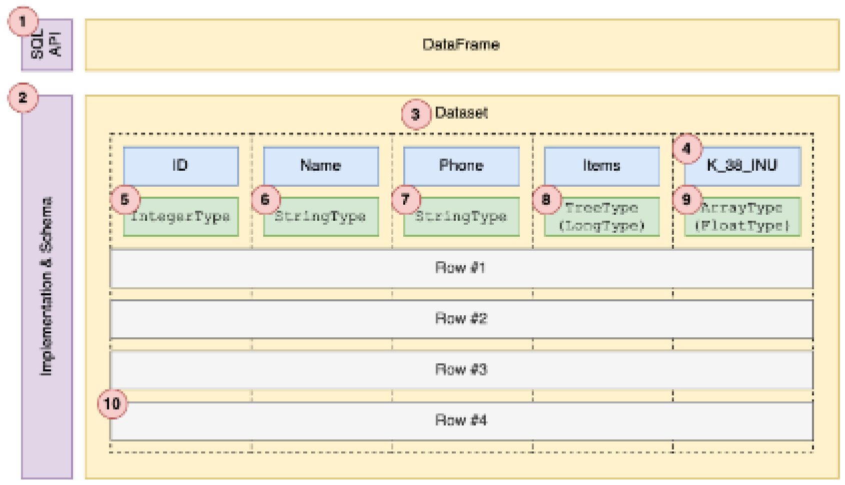
✓ Which of the following statements about executors is correct, assuming that one can consider each of the JVMs working as executors as a pool of task execution slots? 1/1

- ○ Slot is another name for executor.
- ○ There must be less executors than tasks.
- ○ An executor runs on a single core.
- ○ There must be more slots than tasks.
- [✓ Tasks run in parallel via slots.] ✓

✓ Which of the following statements about RDDs is incorrect? 1/1

- [✓ An RDD consists of a single partition.] ✓
- The high-level DataFrame API is built on top of the low-level RDD API.
- RDDs are immutable.
- RDD stands for Resilient Distributed Dataset.
- RDDs are great for precisely instructing Spark on how to do a query.

Which of the elements that are labeled with a circle and a number contain an error or are misrepresented? 0/1



- 1, 10
- 1, 8
- 10
- [✓ 7, 9, 10] ✗
- 1, 4, 6, 9

Correct answer

- [✓ 1, 8]

✓ Which of the following describes characteristics of the Spark UI? 1/1

- ○ Via the Spark UI, workloads can be manually distributed across executors.
- ○ Via the Spark UI, stage execution speed can be modified.
- ○ The Scheduler tab shows how jobs that are run in parallel by multiple users are distributed across the cluster.
- [✓ There is a place in the Spark UI that shows the property spark.executor.memory.] ✓
- ○ Some of the tabs in the Spark UI are named Jobs, Stages, Storage, DAGs, Executors, and SQL.

✓ Which of the following statements about broadcast variables is correct? 1/1

- ○ Broadcast variables are serialized with every single task.
- ○ Broadcast variables are commonly used for tables that do not fit into memory.
- [✓ Broadcast variables are immutable.] ✓
- ○ Broadcast variables are occasionally dynamically updated on a per-task basis.
- ○ Broadcast variables are local to the worker node and not shared across the cluster.

✓ Which of the following is a viable way to improve Spark's performance when dealing with large amounts of data, given that there is only a single application running on the cluster? 1/1

- [✓ Increase values for the properties spark.default.parallelism and spark.sql.shuffle.partitions] ✓
- Decrease values for the properties spark.default.parallelism and spark.sql.partitions
- Increase values for the properties spark.sql.parallelism and spark.sql.partitions
- Increase values for the properties spark.sql.parallelism and spark.sql.shuffle.partitions
- Increase values for the properties spark.dynamicAllocation.maxExecutors, spark.default.parallelism, and spark.sql.shuffle.partitions

✓ Which of the following describes a shuffle? 1/1

- A shuffle is a process that is executed during a broadcast hash join.
- A shuffle is a process that compares data across executors.
- [✓ A shuffle is a process that compares data across partitions.] ✓
- A shuffle is a Spark operation that results from DataFrame.coalesce().
- A shuffle is a process that allocates partitions to executors.

✗ Which of the following describes Spark's Adaptive Query Execution? 0/1

- [●] Adaptive Query Execution features include dynamically coalescing shuffle partitions, dynamically injecting scan filters, and dynamically optimizing skew joins. ✗
- ○ Adaptive Query Execution is enabled in Spark by default.
- ○ Adaptive Query Execution reoptimizes queries at execution points.
- ○ Adaptive Query Execution features are dynamically switching join strategies and dynamically optimizing skew joins.
- ○ Adaptive Query Execution applies to all kinds of queries.

Correct answer

- ● Adaptive Query Execution features are dynamically switching join strategies and dynamically optimizing skew joins.

✓ The code block displayed below contains an error. The code block is intended to join DataFrame itemsDf with the larger DataFrame transactionsDf on column itemId. Find the error. Code block: transactionsDf.join(itemsDf, "itemId", how="broadcast") 1/1

- ○ The syntax is wrong, how= should be removed from the code block.
- ○ The join method should be replaced by the broadcast method.
- ○ Spark will only perform the broadcast operation if this behavior has been enabled on the Spark cluster.
- ○ The larger DataFrame transactionsDf is being broadcasted, rather than the smaller DataFrame itemsDf.
- ● broadcast is not a valid join type. ✓

✓ Which of the following code blocks efficiently converts DataFrame transactionsDf from 12 into 24 partitions? 1/1

- ○ transactionsDf.repartition(24, boost=True)
- ○ transactionsDf.repartition()
- ○ transactionsDf.repartition("itemId", 24)
- ○ transactionsDf.coalesce(24)
- ● transactionsDf.repartition(24) ✓

✓ Which of the following code blocks removes all rows in the 6-column DataFrame transactionsDf that have missing data in at least 3 columns? 1/1

- ○ transactionsDf.dropna("any")
- ● transactionsDf.dropna(thresh=4) ✓
- ○ transactionsDf.drop.na("",2)
- ○ transactionsDf.dropna(thresh=2)
- ○ transactionsDf.dropna("",4)

✓ Which of the following code blocks can be used to save DataFrame transactionsDf to memory only, recalculating partitions that do not fit in memory when they are needed? 1/1

- ○ 1.from pyspark import StorageLevel 2. transactionsDf.cache(StorageLevel.MEMORY_ONLY)
- ○ transactionsDf.cache()
- ○ transactionsDf.storage_level('MEMORY_ONLY')
- ○ transactionsDf.persist()
- ○ transactionsDf.clear_persist()
- [✓ 1. from pyspark import StorageLevel 2.transactionsDf.persist(StorageLevel.MEMORY_ONLY)] ✓

✗ 0/1

The code block displayed below contains an error. The code block should create DataFrame `itemsAttributesDf` which has columns `itemId` and `attribute` and lists every attribute from the `attributes` column in DataFrame `itemsDf` next to the `itemId` of the respective row in `itemsDf`. Find the error.

A sample of DataFrame `itemsDf` is below.

```
+------+----------------------------+-------------------+
|itemId|attributes                  |supplier           |
+------+----------------------------+-------------------+
|1     |[blue, winter, cozy]        |Sports Company Inc.|
|2     |[red, summer, fresh, cooling]|YetiX             |
|3     |[green, summer, travel]     |Sports Company Inc.|
+------+----------------------------+-------------------+
```

Code block:

```
itemsAttributesDf = itemsDf.explode("attributes").alias("attribute").select("attribute", "itemId")
```

○ Since itemId is the index, it does not need to be an argument to the select() method.

○ The alias() method needs to be called after the select() method.

● The explode() method expects a Column object rather than a string. ✗

○ explode() is not a method of DataFrame. explode() should be used inside the select() method instead.

○ The split() method should be used inside the select() method instead of the explode() method.

Correct answer

● explode() is not a method of DataFrame. explode() should be used inside the select() method instead.

✓ Which of the following code blocks reads in parquet file /FileStore/imports.parquet as a DataFrame? 1/1

- ○ spark.mode("parquet").read("/FileStore/imports.parquet")
- ○ spark.read.path("/FileStore/imports.parquet", source="parquet")
- ○ spark.read().parquet("/FileStore/imports.parquet")
- ● spark.read.parquet("/FileStore/imports.parquet") ✓
- ○ spark.read().format('parquet').open("/FileStore/imports.parquet")

✓ 24. The code block shown below should convert up to 5 rows in DataFrame transactionsDf that have the value 25 in column storeId into a Python list. Choose the answer that correctly fills the blanks in the code block to accomplish this. Code block: transactionsDf.__1__(__2__).__3__(__4__) 1/1

- ○ 1. filter 2. "storeId"==25 3. collect 4. 5
- ○ 1. filter 2. col("storeId")==25 3. toLocalIterator 4. 5
- ○ 1. select 2. storeId==25 3. head 4. 5
- ● 1. filter 2. col("storeId")==25 3. take 4. 5 ✓
- ○ 1. filter 2. col("storeId")==25 3. collect 4. 5

✓ Which of the following code blocks reads JSON file imports.json into a DataFrame? 1/1

- ○ spark.read().mode("json").path("/FileStore/imports.json")
- ○ spark.read.format("json").path("/FileStore/imports.json")
- ○ spark.read("json", "/FileStore/imports.json")
- [✓ spark.read.json("/FileStore/imports.json")] ✓
- ○ spark.read().json("/FileStore/imports.json")

✓ Which of the following code blocks returns a DataFrame that has all columns of DataFrame transactionsDf and an additional column predErrorSquared which is the squared value of column predError in DataFrame transactionsDf? 1/1

- ○ transactionsDf.withColumn("predError", pow(col("predErrorSquared"), 2))
- ○ transactionsDf.withColumnRenamed("predErrorSquared", pow(predError, 2))
- [✓ transactionsDf.withColumn("predErrorSquared", pow(col("predError"), lit(2)))] ✓
- ○ transactionsDf.withColumn("predErrorSquared", pow(predError, lit(2)))
- ○ transactionsDf.withColumn("predErrorSquared", "predError"**2)

✓ The code block displayed below contains an error. The code block should return a new DataFrame that only contains rows from DataFrame transactionsDf in which the value in column predError is at least 5. Find the error. Code block:transactionsDf.where("col(predError) >= 5") 1/1

- [✓ ●] The argument to the where method should be "predError >= 5". ✓
- ○ Instead of where(), filter() should be used.
- ○ The expression returns the original DataFrame transactionsDf and not a new DataFrame. To avoid this, the code block should be transactionsDf.toNewDataFrame().where("col(predError) >= 5").
- ○ The argument to the where method cannot be a string.
- ○ Instead of >=, the SQL operator GEQ should be used.

✓ Which of the following code blocks saves DataFrame transactionsDf in location /FileStore/transactions.csv as a CSV file and throws an error if a file already exists in the location? 1/1

- ○ transactionsDf.write.save("/FileStore/transactions.csv")
- ○ transactionsDf.write.format("csv").mode("error").path("/FileStore/transactions.csv")
- ○ transactionsDf.write.format("csv").mode("ignore").path("/FileStore/transactions.csv")
- ○ transactionsDf.write("csv").mode("error").save("/FileStore/transactions.csv")
- [✓ ●] transactionsDf.write.format("csv").mode("error").save("/FileStore/transactions.csv") ✓

✗ The code block shown below should return a DataFrame with two columns, itemId and col. In this DataFrame, for each element in column attributes of DataFrame itemDf there should be a separate row in which the column itemId contains the associated itemId from DataFrame itemsDf. The new DataFrame should only contain rows for rows in DataFrame itemsDf in which the column attributes contains the element cozy. 0/1

A sample of DataFrame `itemsDf` is below.

```
+------+----------------------------+-------------------+
|itemId|attributes                  |supplier           |
+------+----------------------------+-------------------+
|1     |[blue, winter, cozy]        |Sports Company Inc.|
|2     |[red, summer, fresh, cooling]|YetiX             |
|3     |[green, summer, travel]     |Sports Company Inc.|
+------+----------------------------+-------------------+
```

Code block:

```
itemsDf.__1__(__2__).__3__(__4__, __5__(__6__))
```

- ○ 1. filter 2. array_contains("cozy") 3. select 4. "itemId" 5. explode 6. "attributes"
- ○ 1. where 2. "array_contains(attributes, 'cozy')" 3. select 4. itemId 5. explode 6. attributes
- ○ 1. filter 2. "array_contains(attributes, 'cozy')" 3. select 4. "itemId" 5. map 6. "attributes"
- ● 1. filter 2. "array_contains(attributes, cozy)" 3. select 4. "itemId" 5. explode 6. "attributes" ✗
- ○ 1. filter 2. "array_contains(attributes, 'cozy')" 3. select 4. "itemId" 5. explode 6. "attributes"

Correct answer

- ● 1. filter 2. "array_contains(attributes, 'cozy')" 3. select 4. "itemId" 5. explode 6. "attributes"

✓ The code block displayed below contains an error. The code block should return the average of rows in column value grouped by unique storeId. Find the error.Code block:transactionsDf.agg("storeId").avg("value") 1/1

- ○ Instead of avg("value"), avg(col("value")) should be used.
- ○ The avg("value") should be specified as a second argument to agg() instead of being appended to it.
- ○ All column names should be wrapped in col() operators.
- ● agg should be replaced by groupBy. ✓
- ○ "storeId" and "value" should be swapped.

✓ Which of the following code blocks returns a copy of DataFrame itemsDf where the column supplier has been renamed to manufacturer? 1/1

- ○ itemsDf.withColumn(["supplier", "manufacturer"])
- ○ itemsDf.withColumn("supplier").alias("manufacturer")
- ● itemsDf.withColumnRenamed("supplier", "manufacturer") ✓
- ○ itemsDf.withColumnRenamed(col("manufacturer"), col("supplier"))
- ○ itemsDf.withColumnsRenamed("supplier", "manufacturer")

✓ Which of the following code blocks returns DataFrame transactionsDf sorted in descending order by column predError, showing missing values last? 1/1

- ○ transactionsDf.sort(asc_nulls_last("predError"))
- ○ transactionsDf.orderBy("predError").desc_nulls_last()
- ● transactionsDf.sort("predError", ascending=False) ✓
- ○ transactionsDf.desc_nulls_last("predError")
- ○ transactionsDf.orderBy("predError").asc_nulls_last()

✗ The code block displayed below contains an error. The code block is intended to perform an outer join of DataFrames transactionsDf and itemsDf on columns productId and itemId, respectively. Find the error.Code block:transactionsDf.join(itemsDf, [itemsDf.itemId, transactionsDf.productId], "outer") 0/1

- ○ The "outer" argument should be eliminated, since "outer" is the default join type.
- ○ The join type needs to be appended to the join() operator, like join().outer() instead of listing it as the last argument inside the join() call.
- ○ The term [itemsDf.itemId, transactionsDf.productId] should be replaced by itemsDf.itemId == transactionsDf.productId.
- ○ The term [itemsDf.itemId, transactionsDf.productId] should be replaced by itemsDf.col("itemId") == transactionsDf.col("productId").
- ○ The "outer" argument should be eliminated from the call and join should be replaced by joinOuter.

✗ Which of the following code blocks performs a join in which the small DataFrame transactionsDf is sent to all executors where it is joined with DataFrame itemsDf on columns storeId and itemId, respectively? 0/1

- ● itemsDf.join(transactionsDf, itemsDf.itemId == transactionsDf.storeId, "right_outer") ✗
- ○ itemsDf.join(transactionsDf, itemsDf.itemId == transactionsDf.storeId, "broadcast")
- ○ itemsDf.merge(transactionsDf, "itemsDf.itemId == transactionsDf.storeId", "broadcast")
- ○ itemsDf.join(broadcast(transactionsDf), itemsDf.itemId == transactionsDf.storeId)
- ○ itemsDf.join(transactionsDf, broadcast(itemsDf.itemId == transactionsDf.storeId))

Correct answer

- ● itemsDf.join(broadcast(transactionsDf), itemsDf.itemId == transactionsDf.storeId)

✗ Which of the following code blocks reduces a DataFrame from 12 to 6 partitions and performs a full shuffle? 0/1

- ○ DataFrame.repartition(12)
- ○ DataFrame.coalesce(6).shuffle()
- ○ DataFrame.coalesce(6)
- ● DataFrame.coalesce(6, shuffle=True) ✗
- ○ DataFrame.repartition(6)

Correct answer

- ● DataFrame.repartition(6)

✗ The code block displayed below contains an error. The code block is intended to write DataFrame transactionsDf to disk as a parquet file in location /FileStore/transactions_split, using column storeId as key for partitioning. Find the error.Code block:transactionsDf.write.format("parquet").partitionOn("storeId").save("/FileStore/transactions_split")

0/1

- [✓ The format("parquet") expression is inappropriate to use here, "parquet" should be passed as first argument to the save() operator and "/FileStore/transactions_split" as the second argument.] ✗
- Partitioning data by storeId is possible with the partitionBy expression, so partitionOn should be replaced by partitionBy.
- Partitioning data by storeId is possible with the bucketBy expression, so partitionOn should be replaced by bucketBy.
- partitionOn("storeId") should be called before the write operation.
- The format("parquet") expression should be removed and instead, the information should be added to the write expression like so: write("parquet").

Correct answer

- [✓ Partitioning data by storeId is possible with the partitionBy expression, so partitionOn should be replaced by partitionBy.]

The code block displayed below contains an error. The code block is intended to return all columns of DataFrame transactionsDf except for columns predError, productId, and value. Find the error. 0/1

Excerpt of DataFrame `transactionsDf`:

```
+-------------+---------+-----+-------+---------+----+
|transactionId|predError|value|storeId|productId|   f|
+-------------+---------+-----+-------+---------+----+
|            1|        3|    4|     25|        1|null|
|            2|        6|    7|      2|        2|null|
|            3|        3| null|     25|        3|null|
+-------------+---------+-----+-------+---------+----+
```

Code block:

```
transactionsDf.select(~col("predError"), ~col("productId"), ~col("value"))
```

- The select operator should be replaced by the drop operator and the arguments to the drop operator should be column names predError, productId and value wrapped in the col operator so they should be expressed like drop(col(predError), col(productId), col(value)).
- The select operator should be replaced with the deselect operator.
- [✓ The column names in the select operator should not be strings and wrapped in the col operator, so they should be expressed like select(~col(predError), ~col(productId), ~col(value)).] ✗
- The select operator should be replaced by the drop operator.
- The select operator should be replaced by the drop operator and the arguments to the drop operator should be column names predError, productId and value as strings.

Correct answer

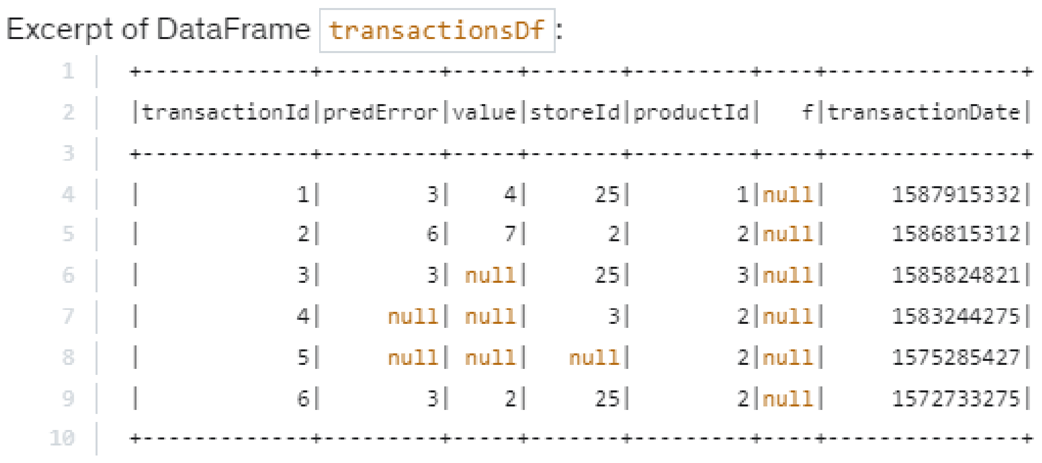
- [✓ The select operator should be replaced by the drop operator and the arguments to the drop operator should be column names predError, productId and value as strings.]

✓ The code block displayed below contains an error. The code block should return DataFrame transactionsDf, but with the column storeId renamed to storeNumber. Find the error.Code block:transactionsDf.withColumn("storeNumber", "storeId") 1/1

- ○ Instead of withColumn, the withColumnRenamed method should be used.
- ○ Arguments "storeNumber" and "storeId" each need to be wrapped in a col() operator.
- ○ Argument "storeId" should be the first and argument "storeNumber" should be the second argument to the withColumn method.
- ○ The withColumn operator should be replaced with the copyDataFrame operator.
- ● Instead of withColumn, the withColumnRenamed method should be used and argument "storeId" should be the first and argument "storeNumber" should be the second argument to that method. ✓

✓ Which of the following code blocks returns a DataFrame with an added column to DataFrame transactionsDf that shows the unix epoch timestamps in column transactionDate as strings in the format month/day/year in column transactionDateFormatted?

1/1


Excerpt of DataFrame `transactionsDf`:

```
+-------------+---------+-----+-------+---------+----+---------------+
|transactionId|predError|value|storeId|productId|   f|transactionDate|
+-------------+---------+-----+-------+---------+----+---------------+
|            1|        3|    4|     25|        1|null|     1587915332|
|            2|        6|    7|      2|        2|null|     1586815312|
|            3|        3| null|     25|        3|null|     1585824821|
|            4|     null| null|      3|        2|null|     1583244275|
|            5|     null| null|   null|        2|null|     1575285427|
|            6|        3|    2|     25|        2|null|     1572733275|
+-------------+---------+-----+-------+---------+----+---------------+
```

- transactionsDf.withColumn("transactionDateFormatted", from_unixtime("transactionDate", format="dd/MM/yyyy"))
- transactionsDf.withColumnRenamed("transactionDate", "transactionDateFormatted", from_unixtime("transactionDateFormatted", format="MM/dd/yyyy"))
- transactionsDf.apply(from_unixtime(format="MM/dd/yyyy")).asColumn("transactionDateFormatted")
- [✓ transactionsDf.withColumn("transactionDateFormatted", from_unixtime("transactionDate", format="MM/dd/yyyy"))] ✓
- transactionsDf.withColumn("transactionDateFormatted", from_unixtime("transactionDate"))

✗ The code block displayed below contains an error. When the code block below has executed, it should have divided DataFrame transactionsDf into 14 parts, based on columns storeId and transactionDate (in this order). Find the error.Code block:transactionsDf.coalesce(14, ("storeId", "transactionDate")) 0/1

- ○ The parentheses around the column names need to be removed and .select() needs to be appended to the code block.
- ○ Operator coalesce needs to be replaced by repartition, the parentheses around the column names need to be removed, and .count() needs to be appended to the code block.
- [✓ Operator coalesce needs to be replaced by repartition, the parentheses around the column names need to be removed, and .select() needs to be appended to the code block.] ✗
- ○ Operator coalesce needs to be replaced by repartition and the parentheses around the column names need to be replaced by square brackets.
- ○ Operator coalesce needs to be replaced by repartition.

Correct answer

- [✓ Operator coalesce needs to be replaced by repartition, the parentheses around the column names need to be removed, and .count() needs to be appended to the code block.]

✓ Which of the following code blocks creates a new DataFrame with two columns season and wind_speed_ms where column season is of data type string and column wind_speed_ms is of data type double? 1/1

- ○ spark.DataFrame({"season": ["winter","summer"], "wind_speed_ms": [4.5, 7.5]})
- [✓ ● spark.createDataFrame([("summer", 4.5), ("winter", 7.5)], ["season", "wind_speed_ms"])] ✓
- ○ 1. from pyspark.sql import types as T 2. spark.createDataFrame((("summer", 4.5), ("winter", 7.5)), T.StructType([T.StructField("season", T.CharType()), T.StructField("season", T.DoubleType())]))
- ○ spark.newDataFrame([("summer", 4.5), ("winter", 7.5)], ["season", "wind_speed_ms"])
- ○ spark.createDataFrame({"season": ["winter","summer"], "wind_speed_ms": [4.5, 7.5]})

✓ The code block shown below should return a column that indicates through boolean variables whether rows in DataFrame transactionsDf have values greater or equal to 20 and smaller or equal to 30 in column storeId and have the value 2 in column productId. Choose the answer that correctly fills the blanks in the code block to accomplish this.transactionsDf.__1__((__2__.__3__) __4__ (__5__)) 1/1

- ○ 1. Select 2. col("storeId")3. between(20, 30)4. And 5. col("productId")==2
- ○ 1. Where 2. col("storeId")3. geq(20).leq(30)4. & 5. col("productId")==2
- ○ 1. Select 2. "storeId" 3. between(20, 30) 4. && 5. col("productId")==2
- ○ 1. Select 2. col("storeId") 3. between(20, 30) 4. && 5. col("productId")=2
- [✓ ● 1. Select 2. col("storeId") 3. between(20, 30) 4. & 5. col("productId")==2] ✓

✓ Which of the following code blocks displays the 10 rows with the smallest values of column value in DataFrame transactionsDf in a nicely formatted way? 1/1

- ○ transactionsDf.sort(asc(value)).show(10)
- [✓ ● transactionsDf.sort(col("value")).show(10)] ✓
- ○ transactionsDf.sort(col("value").desc()).head()
- ○ transactionsDf.sort(col("value").asc()).print(10)
- ○ transactionsDf.orderBy("value").asc().show(10)

✓ Which of the following code blocks uses a schema fileSchema to read a parquet file at location filePath into a DataFrame? 1/1

- [✓ ● spark.read.schema(fileSchema).format("parquet").load(filePath)] ✓
- ○ spark.read.schema("fileSchema").format("parquet").load(filePath)
- ○ spark.read().schema(fileSchema).parquet(filePath)
- ○ spark.read().schema(fileSchema).format(parquet).load(filePath)
- ○ spark.read.schema(fileSchema).open(filePath)

✓ Which of the following code blocks returns only rows from DataFrame transactionsDf in which values in column productId are unique? 1/1

- ○ transactionsDf.distinct("productId")
- [✓ ● transactionsDf.dropDuplicates(subset=["productId"])] ✓
- ○ transactionsDf.drop_duplicates(subset="productId")
- ○ transactionsDf.unique("productId")
- ○ transactionsDf.dropDuplicates(subset="productId")

The code block displayed below contains an error. The code block below is intended to add a column itemNameElements to DataFrame itemsDf that includes an array of all words in column itemName. Find the error. 0/1

Sample of DataFrame `itemsDf`:

```
+------+----------------------------------+-------------------+
|itemId|itemName                          |supplier           |
+------+----------------------------------+-------------------+
|1     |Thick Coat for Walking in the Snow|Sports Company Inc.|
|2     |Elegant Outdoors Summer Dress     |YetiX              |
|3     |Outdoors Backpack                 |Sports Company Inc.|
+------+----------------------------------+-------------------+
```

Code block:

```
itemsDf.withColumnRenamed("itemNameElements", split("itemName"))
```

- [✓ All column names need to be wrapped in the col() operator.] ✗
- Operator withColumnRenamed needs to be replaced with operator withColumn and a second argument "," needs to be passed to the split method.
- Operator withColumnRenamed needs to be replaced with operator withColumn and the split method needs to be replaced by the splitString method.
- Operator withColumnRenamed needs to be replaced with operator withColumn and a second argument " " needs to be passed to the split method.
- The expressions "itemNameElements" and split("itemName") need to be swapped.

Correct answer

- [✓ Operator withColumnRenamed needs to be replaced with operator withColumn and a second argument " " needs to be passed to the split method.]

✓ The code block shown below should return all rows of DataFrame itemsDf that have at least 3 items in column itemNameElements. Choose the answer that correctly fills the blanks in the code block to accomplish this. 1/1

Example of DataFrame `itemsDf`:

```
+------+----------------------------------+-------------------+------------------------------------------+
|itemId|itemName                          |supplier           |itemNameElements                          |
+------+----------------------------------+-------------------+------------------------------------------+
|1     |Thick Coat for Walking in the Snow|Sports Company Inc.|[Thick, Coat, for, Walking, in, the, Snow]|
|2     |Elegant Outdoors Summer Dress     |YetiX              |[Elegant, Outdoors, Summer, Dress]        |
|3     |Outdoors Backpack                 |Sports Company Inc.|[Outdoors, Backpack]                      |
+------+----------------------------------+-------------------+------------------------------------------+
```

Code block:

- ○ 1. Select 2. Count 3. col("itemNameElements") 4. >3
- ○ 1. Filter 2. Count 3. itemNameElements 4. >=3
- ○ 1. Select 2. Count 3. "itemNameElements" 4. >3
- [✓ 1. Filter 2. Size 3. "itemNameElements" 4. >=3] ✓
- ○ 1. Select 2. Size 3. "itemNameElements" 4. >3

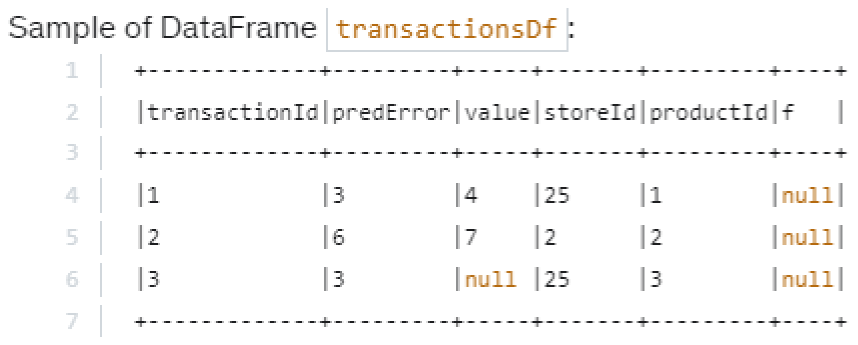
✓ The code block displayed below contains an error. The code block should use Python method find_most_freq_letter to find the letter present most in column itemName of DataFrame itemsDf and return it in a new column most_frequent_letter. Find the error. Code block: 1. find_most_freq_letter_udf = udf(find_most_freq_letter) 2.itemsDf.withColumn("most_frequent_letter", find_most_freq_letter("itemName")) 1/1

- [✓ Spark is not using the UDF method correctly.] ✓
- ○ The UDF method is not registered correctly, since the return type is missing.
- ○ The "itemName" expression should be wrapped in col().
- ○ UDFs do not exist in PySpark.
- ○ Spark is not adding a column.

✓ Which of the following code blocks returns about 150 randomly selected rows from the 1000-row DataFrame transactionsDf, assuming that any row can appear more than once in the returned DataFrame? 1/1

- ○ transactionsDf.resample(0.15, False, 3142)
- ○ transactionsDf.sample(0.15, False, 3142)
- ○ transactionsDf.sample(0.15)
- ○ transactionsDf.sample(0.85, 8429)
- [✓ transactionsDf.sample(True, 0.15, 8261)] ✓

✕ Which of the following code blocks returns a DataFrame where columns predError and productId are removed from DataFrame transactionsDf? 0/1



Sample of DataFrame `transactionsDf`:

```
+-------------+---------+-----+-------+---------+----+
|transactionId|predError|value|storeId|productId|f   |
+-------------+---------+-----+-------+---------+----+
|1            |3        |4    |25     |1        |null|
|2            |6        |7    |2      |2        |null|
|3            |3        |null |25     |3        |null|
+-------------+---------+-----+-------+---------+----+
```

- ○ transactionsDf.withColumnRemoved("predError", "productId")
- [✓ transactionsDf.drop(["predError", "productId", "associateId"])] ✕
- ○ transactionsDf.drop("predError", "productId", "associateId")
- ○ transactionsDf.dropColumns("predError", "productId", "associateId")
- ○ transactionsDf.drop(col("predError", "productId"))

Correct answer

- [✓ transactionsDf.drop("predError", "productId", "associateId")]

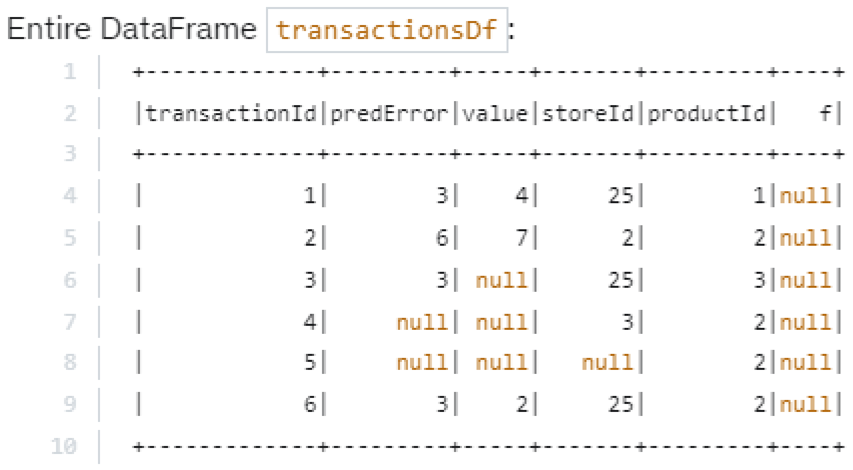
✓ The code block displayed below contains an error. The code block should return a DataFrame where all entries in column supplier contain the letter combination et in this order. Find the error.Code block:itemsDf.filter(Column('supplier').isin('et')) 1/1

- [✓ The Column operator should be replaced by the col operator and instead of isin, contains should be used.] ✓
- The expression inside the filter parenthesis is malformed and should be replaced by isin('et', 'supplier').
- Instead of isin, it should be checked whether column supplier contains the letters et, so isin should be replaced with contains. In addition, the column should be accessed using col['supplier'].
- The expression only returns a single column and filter should be replaced by select.

✓ The code block shown below should write DataFrame transactionsDf to disk at path csvPath as a single CSV file, using tabs (\t characters) as separators between columns, expressing missing values as string n/a, and omitting a header row with column names. Choose the answer that correctly fills the blanks in the code block to accomplish this.transactionsDf.__1__.write.__2__(__3__, "\t").__4__.__5__(csvPath) 1/1

- 1. coalesce(1) 2. Option 3. "Sep" 4. option("header", True)5. path
- 1. coalesce(1) 2. Option 3. "Colsep" 4. option("nullValue", "n/a") 5. path
- [✓ 1. repartition(1)2. Option 3. "Sep" 4. option("nullValue", "n/a") 5. csv] ✓
- 1. Csv 2. Option 3. "Sep" 4. option("emptyValue", "n/a") 5. path
- 1. repartition(1)2. Mode 3. "Sep" 4. mode("nullValue", "n/a") 5. csv

Which of the following code blocks returns a new DataFrame with only columns predError and values of every second row of DataFrame transactionsDf? 0/1



Entire DataFrame `transactionsDf`:

```
+-------------+---------+-----+-------+---------+----+
|transactionId|predError|value|storeId|productId|   f|
+-------------+---------+-----+-------+---------+----+
|            1|        3|    4|     25|        1|null|
|            2|        6|    7|      2|        2|null|
|            3|        3| null|     25|        3|null|
|            4|     null| null|      3|        2|null|
|            5|     null| null|   null|        2|null|
|            6|        3|    2|     25|        2|null|
+-------------+---------+-----+-------+---------+----+
```

[✓ transactionsDf.filter(col("transactionId").isin([3,4,6])).select([predError, value])] ✗

transactionsDf.select(col("transactionId").isin([3,4,6]), "predError", "value")

transactionsDf.filter("transactionId" % 2 == 0).select("predError", "value")

transactionsDf.filter(col("transactionId") % 2 == 0).select("predError", "value")

1.transactionsDf.createOrReplaceTempView("transactionsDf") 2.spark.sql("FROM transactionsDf SELECT predError, value WHERE transactionId % 2 = 2")

transactionsDf.filter(col(transactionId).isin([3,4,6]))

Correct answer

[✓ transactionsDf.filter(col("transactionId") % 2 == 0).select("predError", "value")]

✓ The code block displayed below contains an error. The code block should display the schema of DataFrame transactionsDf. Find the error.Code block:transactionsDf.rdd.printSchema 1/1

- ○ There is no way to print a schema directly in Spark, since the schema can be printed easily through using print(transactionsDf.columns), so that should be used instead.
- ○ The code block should be wrapped into a print() operation.
- ○ printSchema is only accessible through the spark session, so the code block should be rewritten as spark.printSchema(transactionsDf).
- ● printSchema is a method and should be written as printSchema(). It is also not callable through transactionsDf.rdd, but should be called directly from transactionsDf. ✓
- ○ printSchema is a not a method of transactionsDf.rdd. Instead, the schema should be printed via transactionsDf.print_schema().

✓ Which of the following code blocks returns a one-column DataFrame of all values in column supplier of DataFrame itemsDf that do not contain the letter X? In the DataFrame, every value should only be listed once. 1/1

Sample of DataFrame `itemsDf`:

```
+------+--------------------+-------------------+-------------------+
|itemId|            itemName|         attributes|           supplier|
+------+--------------------+-------------------+-------------------+
|     1|Thick Coat for Wa...|[blue, winter, cozy]|Sports Company Inc.|
|     2|Elegant Outdoors ...|[red, summer, fre...|              YetiX|
|     3|   Outdoors Backpack|[green, summer, t...|Sports Company Inc.|
+------+--------------------+-------------------+-------------------+
```

- ○ itemsDf.filter(col(supplier).not_contains('X')).select(supplier).distinct()
- ○ itemsDf.select(~col('supplier').contains('X')).distinct()
- ○ itemsDf.filter(not(col('supplier').contains('X'))).select('supplier').unique()
- ● itemsDf.filter(~col('supplier').contains('X')).select('supplier').distinct() ✓
- ○ itemsDf.filter(!col('supplier').contains('X')).select(col('supplier')).unique()

✗ In which order should the code blocks shown below be run in order to create a table of all values in column attributes next to the respective values in column supplier in DataFrame itemsDf? 0/1

1. `itemsDf.createOrReplaceView("itemsDf")`
2. `spark.sql("SELECT 'supplier', explode('Attributes') FROM itemsDf")`
3. `spark.sql("SELECT supplier, explode(attributes) FROM itemsDf")`
4. `itemsDf.createOrReplaceTempView("itemsDf")`

- ○ 4, 3
- ○ 1, 3
- [✓ 2] ✗
- ○ 4, 2
- ○ 1, 2

Correct answer

[✓ 4, 3]

✓ The code block shown below should return a copy of DataFrame transactionsDf without columns value and productId and with an additional column associateId that has the value 5. Choose the answer that correctly fills the blanks in the code block to accomplish this.transactionsDf.__1__(__2__, __3__).__4__(__5__, 'value') 1/1

- ○ 1. withColumn 2. 'associateId'3. 5 4. Remove 5. 'productId'
- ○ 1. withNewColumn 2. associateId 3. lit(5)4. Drop 5. productId
- [✓ 1. withColumn 2. 'associateId'3. lit(5) 4. Drop 5. 'productId'] ✓
- ○ 1. withColumnRenamed 2. 'associateId'3. 5 4. Drop 5. 'productId'
- ○ 1. withColumn 2. col(associateId) 3. lit(5)4. Drop 5. col(productId)

✓ The code block shown below should write DataFrame transactionsDf as a parquet file to path storeDir, using brotli compression and replacing any previously existing file. Choose the answer that correctly fills the blanks in the code block to accomplish this.transactionsDf.__1__.format("parquet").__2__(__3__).option(__4__, "brotli").__5__(storeDir) 1/1

- ○ 1. Save 2. Mode 3. "Ignore" 4. "Compression" 5. Path
- ○ 1. Store 2. With 3. "Replacement" 4. "Compression" 5. path
- [✓ 1. Write 2. Mode 3. "Overwrite" 4. "Compression" 5. save] ✓
- ○ 1. Save 2. Mode 3. "Replace" 4. "Compression" 5. path
- ○ 1. Write 2. Mode 3. "Overwrite" 4. Compression 5. parquet

✓ Which of the following code blocks will silently skip writing itemsDf in avro format to location fileLocation if there is a file at that location already? 1/1

- ○ itemsDf.write.avro(fileLocation)
- [✓ ● itemsDf.write.format("avro").mode("ignore").save(fileLocation)] ✓
- ○ itemsDf.write.format("avro").mode("errorifexists").save(fileLocation)
- ○ itemsDf.save.format("avro").mode("ignore").write(fileLocation)
- ○ spark.DataFrameWriter(itemsDf).format("avro").write(fileLocation)

✗ The code block displayed below contains at least one error. The code block should return a DataFrame with only one column, result. That column should include all values in column value from DataFrame transactionsDf raised to the power of 5, and a null value for rows in which there is no value in column value. Find the error(s). 0/1

Code block:

```
from pyspark.sql.functions import udf
from pyspark.sql import types as T

transactionsDf.createOrReplaceTempView('transactions')

def pow_5(x):
  return x**5

spark.udf.register(pow_5, 'power_5_udf', T.LongType())
spark.sql('SELECT power_5_udf(value) FROM transactions')
```

○ The pow_5 method is unable to handle empty values in column value and the name of the column in the returned DataFrame is not result.

○ The returned DataFrame includes multiple columns instead of just one column.

● The pow_5 method is unable to handle empty values in column value, the name of the column in the returned DataFrame is not result, and the SparkSession cannot access the transactionsDf DataFrame. ✗

○ The pow_5 method is unable to handle empty values in column value, the name of the column in the returned DataFrame is not result, and Spark driver does not call the UDF function appropriately.

○ The pow_5 method is unable to handle empty values in column value, the UDF function is not registered properly with the Spark driver, and the name of the column in the returned DataFrame is not result.

Correct answer

● The pow_5 method is unable to handle empty values in column value, the UDF function is not registered properly with the Spark driver, and the name of the column in the returned DataFrame is not result.

Google Forms